TCP CUBIC in ns-3

CS577
Brett Levasseur
Outline

• Introduction
• CUBIC
• CUBIC in Linux
• ns-3 Implementation
• Results
• Conclusions
Introduction

- TCP grows cwnd too slowly for large bandwidth connections
- New TCP Variant needed

Graph showing:
- 10Gbps
- Almost 1.4 hours to take up ½ possible cwnd
- 1,250 byte packets
- ssthresh ~ 8000 bytes
- 100ms RTT
CUBIC

- BIC was first attempt
- CUBIC simplified and improved upon BIC
- Grow cwnd slower around loss events
CUBIC Basics

• cwnd growth

\[ W(t) = C(t - K)^3 + W_{\text{max}} \]

• Packet loss

\[ K = \sqrt[3]{\frac{W_{\text{max}}}{C}} \frac{1}{\beta} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>CUBIC parameter</td>
</tr>
<tr>
<td>t</td>
<td>Elapsed time from the last window reduction</td>
</tr>
<tr>
<td>K</td>
<td>Time period to increase W to Wmax</td>
</tr>
<tr>
<td>W</td>
<td>Current cwnd</td>
</tr>
<tr>
<td>W_{\text{max}}</td>
<td>cwnd at last window reduction</td>
</tr>
<tr>
<td>\beta</td>
<td>Window decrease constant</td>
</tr>
</tbody>
</table>
• At loss event set $W_{\text{max}}$, reduce $\text{cwnd}$ by $\beta$ and calculate $K$
CUBIC Basics

- cwnd grows back to K when \( t = K \)
CUBIC in Linux

• Not implemented as in the CUBIC paper
• cwnd grows in increments of segment sizes
• Custom method for calculating cube roots
• Checks for error conditions
• Unit scaling
Growing cwnd

- Linux only grows cwnd by full segments
- CUBIC can grow cwnd less than full segment
- Same impact by increasing amount of time between updates
Scaling in CUBIC

• Most scaling is related to time
• Variable ‘t’ measured with TCP timestamps
  – Timestamps use clock cycles to increment
  – Units are called jiffies in the Linux Kernel
• Number of milliseconds in a jiffy depends on the CPU’s clock
• Scaling required to get time units correct
ns-3 Implementation

• Object oriented design
• Generic TCP defined
• TCP variants are extended from base
• TCP headers and buffers provided
• Added TcpCubic object
  – tcp-cubic.cc
  – tcp-cubic.h
ns-3 Methods

• NewAck – called for every new ACK received
  – Normal cwnd updates in slow start
  – CUBIC updates otherwise

• DupAck – called for every duplicate ACK received
  – Normal operation when < 3 duplicates
  – For 3 duplicate ACKs reduce cwnd
CUBIC Methods

• CubicRoot – Find the cubic root of a number
  – Based on Linux Kernel implementation
• CubicUpdate – Calculate the cwnd target for CUBIC
• CubicTcpFriendliness – Change the cwnd target for TCP Friendliness
• CubicReset – Reset CUBIC parameters
CUBIC Flow

If outside slow start

Recommended cwnd growth

Set K for loss event

Check TCP Friendliness

Recommended cwnd growth
Issues

• ns-3 does not have TCP timestamps
• Simulation clock used instead
• Requires adjustments to calculating ‘t’ due to different units
• Could remove the use of jiffy code but much of the Linux implementation relies on scaling factors based on the system clock
Results

• Compare to real world CUBIC example

• Examine simulation results
  – Verify cwnd reduction
  – Verify cwnd growth in relation to Wmax

• Compare simulated CUBIC to simulated NewReno
Simulation Scenario

- Simple sender and sink topology
- Packet sizes 536 bytes
- Transmission rate 1Mbps
- Delay 40ms
- Error rate – Causes lost packets at the receiver
• Measurement and simulation have similar CUBIC curve

• Number of segments similar
Packet Loss

- Before loss $cwnd = 216$
- After loss $cwnd = 172$
- $\beta = 819$
- $\text{BICTCP\_BETA\_SCALE} = 1024$

\[
cwnd = ssthresh = \max\left(\frac{cwnd \times \beta}{\text{BICTCP\_BETA\_SCALE}}, 2\right)
\]

\[
172.76 = \max\left(\frac{216 \times 819}{1024}, 2\right)
\]
• Before and after additional scaling of ‘t’
• More work is needed for using simulator clock with ‘t’
NewReno Comparison

• Same simulation run with CUBIC and NewReno
• Both increment the same under slow start
• CUBIC grows cwnd faster
• CUBIC handles packet loss better than NewReno
Conclusions

• Created a CUBIC implementation in ns-3
• Similar cwnd growth to actual CUBIC measurements
• Current version outperforms NewReno
• Scaling adjustments required
Questions